Amendments to the Claims

Please amend claims to be as follows.

1. (currently amended) A method of compiling a computer program, the method comprising:

receiving a plurality of modules of source code;

generating intermediate representations corresponding to the modules;

extracting a set of data from the intermediate representations to create an inliner summary for each module;

using the inliner summaries and a globally-sorted working-list based order in an inline analysis phase, without using the intermediate representations in the inline analysis phase, to determine which call sites in the modules are to be inlined by substituting code from a called module, wherein said globally-sorted working-list based order is dynamically updated during the inline analysis phase; and

after a call site is determined to be inlined, updating a call graph of the routines and call sites, and updating the inliner summaries throughout the call graph,

wherein determining the call sites to be inlined involves proceeding only once through the call sites in said dynamically-updated globally-sorted working-list based order

2. (canceled)

3. (previously presented) The method of claim 1, further comprising, after the call graph and inliner summaries are updated,

re-calculating profitabilities associated with remaining call sites; and

re-ordering the working list using the re-calculated profitabilities.

4. (previously presented) The method of claim 3, wherein updating the inliner summaries comprises determining nodes and edges of the call graph that are affected by the inlining of the call site and updating those inliner summaries corresponding to the affected nodes and edges.

5. (previously presented) The method of claim 4, wherein the edge summaries include at least a call site execution count and a signature type.

6. (previously presented) The method of claim 4, wherein the node summaries include at least a code size, a routine execution count, and a call-graph height.

7. (original) The method of claim 1, wherein the inline analysis phase is separate and distinct from an inline transformation phase.

8. (currently amended) An apparatus for compiling a computer program, the apparatus comprising:

a processor configured to execute computer-readable code;

a memory system configured to store data;

computer-readable code for a front-end portion of a compiler program, the front-end portion of the compiler program being configured to receive a plurality of modules of source code, generate intermediate representations corresponding to the modules, and extract a set of data from the intermediate representations to generate inliner summaries for the modules; and

computer-readable code for a cross-module optimizer of the compiler program, the cross-module optimizer being configured to use the inliner summaries and a dynamically-updated globally-sorted working-list based order to analyze the call sites in an inline analysis phase, without using the intermediate representations, so as to determine which call sites in the modules are to be inlined by substituting code from a called module, and to update a call graph

of the routines and call sites, and to update the inliner summaries, after a call site is determined to be inlined, wherein determining the call sites to be inlined involves proceeding only once through the call sites in said dynamically-updated globally-sorted working-list based order.

9. (canceled)

10. (previously presented) The apparatus of claim 8, wherein the cross-module optimizer is further configured to re-calculate profitabilities associated with remaining call sites, and re-order the working list using the re-calculated profitabilities, after the call graph and inliner summaries are updated.

11. (previously presented) The apparatus of claim 10 wherein the cross-module optimizer is further configured to update the inliner summaries by determining nodes and edges of the call graph that are affected by the inlining of the call site and update those inliner summaries corresponding to the affected nodes and edges.

12. (previously presented) The apparatus of claim 11, wherein the edge summaries generated by the front-end portion include at least a call site execution count and a signature type.

13. (previously presented) The apparatus of claim 11, wherein the node summaries generated by the front-end portion include at least a code size, a routine execution count, and a call-graph height.

14. (previously presented) The apparatus of claim 8, wherein the cross-module optimizer is further configured to perform the inline analysis phase separately and distinctly from an inline transformation phase.

15. (currently amended) A computer program product comprising a computer-usable medium having computer-readable code embodied therein, the computer program product being compiled from a plurality of modules of source code using inliner summaries and a dynamically-updated globally-sorted working-list based order in an inline analysis phase, without using intermediate representations, to determine which call sites in the modules are to be inlined by substituting code from a called module and, after a call site is determined to be inlined, to determine a call graph of the routines and call sites, and to update the inliner summaries throughout the call graph, wherein determining the call sites to be inlined involves proceeding only once through the call sites in said dynamically-updated globally-sorted working-list based order.

16. (currently amended) A method of compiling a computer program with a plurality of modules of source code and corresponding intermediate representations, the method comprising:

extracting a set of data from the intermediate representations of the modules to create [[a light-weight]] an inliner summary for each module;

using the inliner summaries in a one-pass inline analysis phase, without using the intermediate representations, to determine which call sites to inline, in what order to inline them, and preserving a same order during the transformation phase;

formulating a measure of goodness for each call site from the [[light-weight]] inliner summary;

using a technique to dynamically update information in the [[light-weight]] summary for potentially all call-graph nodes and edges every time a call site is accepted for inlining; and

using a globally sorted work-list and an associated table to maintain and manipulate the call sites and dynamically updating the work-list every time a call site is accepted for inlining.

17. (currently amended) The method of claim 16, wherein the inliner summaries are comprised of: a code size; a call site profile count; a critical path length; an execution time; a node level; ~~a call savings; an optimization savings;~~ a level criticality; ~~a normalized level;~~ and a total execution count.

18. (previously presented) The method of claim 16, wherein an arbitrary inlining order among the call sites in the call graph is selectable in an inline analysis phase, and wherein that same order is followed in an inline transformation phase.

19. (previously presented) The method of claim 16, wherein a measure of goodness for each call site is computed from the light-weight inliner summary, and a total profit is computed as a product of component profit factors, and a total cost is computed as a product of component cost factors.

20. (previously presented) The method of claim 16, wherein information in the light-weight inliner summary corresponding to call-graph nodes and edges are updated each time a call site is accepted for inlining, and wherein a goodness factor of each call site with updated summary information is re-computed.

21. (previously presented) The method of claim 16, wherein a globally-sorted work list and an associated table are maintained and used to continuously order the work list and extract a call site with a highest goodness factor.